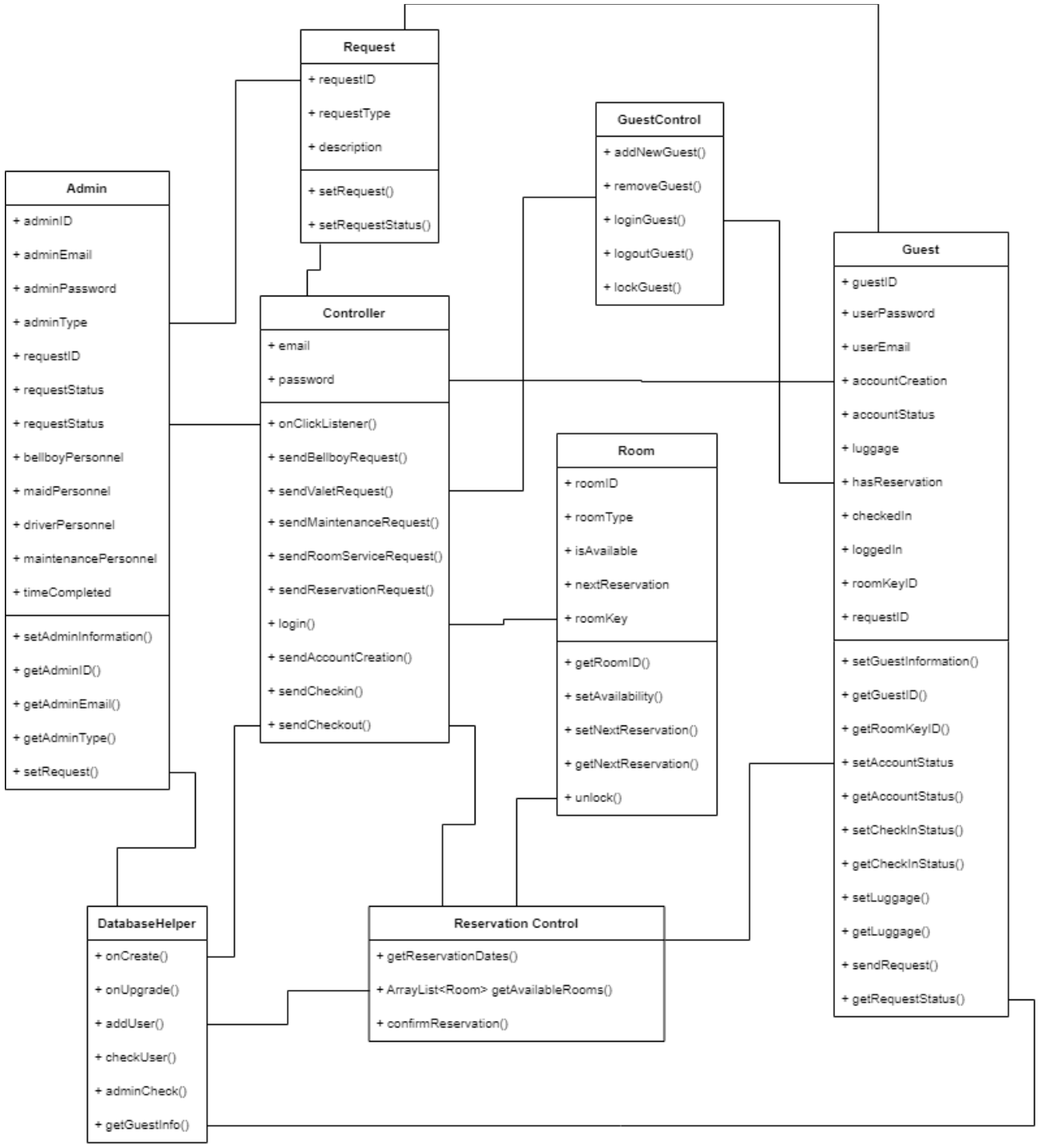
RU Staying

Software Engineering
01:332:452
Report 2: Part 2

By Group #11
Keya Patel
Zain Sayed
Mohammed Sapin
Purna Haque
Nga Man (Mandy) Cheng
Rameen Masood
Shilp Shah
Mathew Varghese
Thomas Tran
Eric Zhang

Github: https://github.com/mohammedsapin/RUStaying
Submission Date: March 10, 2019

Class Diagrams and Interface Specification



Classes

Controller

The controller coordinates actions of all concepts associated with the use cases and logically groups the subsystems to work together. It is the main interface between the system and the application that the user sees.
The controller will be handling all user interactions with the app, such as button clicks and entering text information. It is also responsible for transitioning XML pages as the app is event-controlled and the user can navigate to multiple pages.

Methods:

1) On-click listeners for all buttons
    a) Will trigger different pages in the UI and button clicks may cause any of the below methods to be called
2) sendBellboyRequest()
    a) Will call the Hotel Services Controller
3) sendValetRequest()
    a) Will call the Hotel Services Controller
4) sendMaintenanceRequest()
    a) Will call the Hotel Services Controller
5) sendRoomServiceRequest()
    a) Will call the Hotel Services Controller
6) sendReservationRequest()
    a) Will call the Reservation Controller
7) login(string email, string password)
    a) Will call the Guest Controller and allow access to rest of app
8) sendAccountCreation()
    a) Will call the Guest Controller and allow user to login
9) sendCheckin()
    a) Will call the Guest Controller
10) sendCheckout()
    a) Will call the Guest Controller

Guest

Attributes:

1) int guestId
    a) Unique ID number given to each guest
2) String userPassword

    a) Specific password of guest to login
3) String userEmail
    a) Specific email address of guest to login
4) Date accountCreation
    a) Stores when exactly the account was created
5) String accountStatus
    a) Determines the status of the account (active, inactive, locked etc)
6) int luggage
    a) Determines whether the guest has specified if they have luggage or not
    b) Information used by Bellboy service
7) boolean hasReservation
    a) Boolean value to see if guest has made a reservation for a room, in which case they may check in upon arrival
8) boolean checkedIn
    a) A boolean value to see if guest is checked in or not
    b) Checked-in status confirms that the guest is at the hotel currently
    c) Going from a checked-in equals "true" status to a checked-in equals "false" status indicates that the guest has finished their stay and have now checked out
9) boolean loggedIn
    a) A boolean value to see if the guest is logged into their account
10) String roomKeyID
    a) Integer value with a unique room key ID number for guests to unlock their room doors
11) int requestId
    a) Unique ID number pertaining to a new request a guest has made

Methods:

1) void setGuestInformation(int guestId, String userEmail, String userPassword)
    a) Creates new guest ID
    b) Stores new guest's entered email and password
    c) Creates roomKeyID variable for guest's room
2) String getGuestId()
    a) Get guest ID based on guest information
3) String getRoomKeyID()
    a) Get guest room key information based on guest information
4) void setAccountStatus(String status, boolean loggedIn)
    a) Sets account status of account corresponding to guest ID (active, inactive, locked, etc)
    b) Sets loggedIn variable of account
5) String getAccountStatus()

    a) Gets account status of account corresponding to guest ID (if user is logged in, if account is active, inactive, or locked)
6) void setCheckedInStatus(boolean hasReservation)
    a) Checks hasReservation variable, then sets checkedIn variable
7) Boolean getCheckedInStatus()
    a) Gets check-in status of account corresponding to guest ID
8) void setLuggage(int luggage)
    a) Sets number of luggage pieces user is bringing
9) int getLuggage()
    a) Gets number of luggage pieces user is bringing
10) int sendRequest(Guest guestId)
    a) Creates a new request ID
    b) Sends the request ID to administrative staff containing information of the guest who sent the request
    c) Returns request ID for guest to check on updates pertaining to their request
11) String getRequestStatus(requestId)
    a) Returns the request status of "completed", "in progress", or "processing" based on request ID

GuestControl - Interacts directly with main controller for functionality

Methods

1) addNewGuest()
    a) Add new guest to database
2) removeGuest(int guestId)
    a) Remove a guest from database, specified by ID number
3) loginGuest(int guestId)
    a) Mark boolean value to true if guest is successfully logged in
4) logoutGuest(guestId)
    a) Mark boolean value to false if guest logs out of their account
5) lockGuest(guestId)
    a) Mark boolean value to true if guest has been locked from their account

RequestObject

Attributes:

1) Int requestId
    a) Identifying number of a new request
2) String requestType
    a) String value that contains the type of request a guest has made ("bellboy", "maintenance", "room service", or "travel")

3) String description
    a) String value containing a description of the guest's problems or information/specifics of the request

Methods:

1) Void setRequest(String requestType)
    a) Sets the type of request to one of "bellboy", "maintenance", "room service", or "travel"
    b) Sets details of request (i.e. description of problem or specifications)
    c) Creates a new request ID for this request, the number of the request will depend on the type of request, for example, if the user has requested a bellyboy, then the first integer of the request ID will begin with a 1, if maintenance, then the first integer of the request ID will begin with a 2, etc.
2) void setRequestStatus(requestId)
    a) Sets status of request to "completed", "in progress", or "processing"

Admin

Attributes:

1) int adminId
    a) Unique ID number given to a member of the administrative staff
2) String adminEmail
    a) Email of an administrative staff to login
3) String adminPassword
    a) Specific password of staff to login
4) String adminType
    a) Specify the exact role of Admin
    b) "Bellboy", "Maid", "Manager", "Driver", "Maintenance"
5) int requestId
    a) Contains request Id
6) String requestStatus
    a) Contains status of request as "processing", "in progress", or "completed"
7) int bellboyPersonnel
    a) Number of available bellboys to perform bellboy requests
        i) If 0 requestStatus remains "processing"
        ii) If >0 requestStatus becomes "in progress"
8) int maidPersonnel
    a) Number of available maids to perform room service requests
        i) If 0 requestStatus remains "processing"
        ii) If >0 requestStatus becomes "in progress"

9) int driverPersonnel
    a) Number of available valet drivers to perform valet & travel requests
        i) If 0 requestStatus remains “processing”
        ii) If >0 requestStatus becomes “in progress”
10) int maintenencePersonnel
    a) Number of available members of maintenance to perform maintenance requests
        i) If 0 request remains “processing”
        ii) If >0 requests becomes “in progress”
11) int timeCompleted
    a) Contains time needed for a request to be completed. A staff member who has received a request will have a timeCompleted integer assigned to them. A clock will determine if the value stored in timeCompleted since the request started has been reached, in which case timeCompleted will be set to 0 and the available personnel variable associated for the type of staff member will be incremented indicating that the staff member has completed their task and is now available for another. The request status will also then be updated to “completed”

Methods:

1) void setAdminInformation(String adminEmail, String adminPassword, String adminType)
    a) Set admin entered email
    b) Set admin entered password
    c) Set admin entered type
    d) Create admin’s new ID
2) int getAdminId()
    a) Gets admin’s ID
3) String get adminEmail()
    a) Gets admin’s email
4) String getAdminType()
    a) Get admin’s type
5) String setRequest(requestId)
    a) Checks number of available personnel depending on who the request was made to
    b) Sets status of “processing”, “in progress”, or “completed” depending on the available personnel variable values. If no personnel are available, this method will be put in a loop to check every minute if a staff member has become available. If one has become available, sets status from “processing” to “in progress”

Room Object

Attributes:

1) int roomId
    a) Unique number assigned to each room
2) String roomType
    a) “Single”, “Double”, “Queen”, or “King”
3) boolean isAvailable
    a) Flag to indicate if the room is occupied or not
4) Date nextReservation
    a) Date object to indicate when the room is reserved if it is not currently in use
5) String roomKey
    a) Unique room key string given to each guest staying in a specific room, or to staff to enter the room

Methods:

1) int getRoomId()
    a) Returns the unique number of the room
2) void setAvailability(boolean isAvailable)
    a) Method to set the availability to of the room to true or false
3) void setNextReservation(Date nextReservation)
    a) Method to set the reservation of the room if a new request was made successfully
4) Date getNextReservation()
    a) Method to get the next reservation date of the room
5) boolean unlock(int roomKey)
    a) Method to compare if user entered room key matches with the room’s actual key value, will return true if unlocked and false if the key do not match, leaving the door locked

ReservationControl - Implements functionality to create new reservations

Methods

1) getReservationDates()
    a) Gets information entered by guest in the UI
2) ArrayList<Room> getAvailableRooms(Date startDate, Date endDate)
    a) Based on the duration of stay, check all the rooms that are available and return an ArrayList of these Room objects
    b) The controller will output this ArrayList to the UI so the guest can see the available rooms
3) confirmReservation(Room bookedRoom)
    a) When the guest selects a room, send that object back to mark the room as reserved for those dates

DatabaseHelper

Main object that manages the database objects and all the methods that directly input and extract data from tables. As we continue implementing functionality, the DatabaseHelper methods will continue to increase. Maintaining database information is the backbone of our app because all the data is used to help serve the guest.

Methods

1) onCreate()
    a) Default by Android studio
2) onUpgrade()
    a) Default by Android studio
3) addUser()
    a) Adds a new user to the table
4) checkUser()
    a) Checks if a user exists in the table
5) adminCheck()
    a) Checks if an admin login is valid.
6) getGuestInfo(int guestId)
    a) Returns a Guest object from the given ID number

Traceability Matrix

	Domain Concepts							
Class	Controller	UserVerification	UserManagement	RoomControl	HotelServices	RoomTracking	HotelTracking	AdminControl
Controller	x							
Guest		x	x			x		
GuestControl		x	x			x		
Admin				x	x	x	x	x
RoomObject				x		x		
ReservationControl				x		x		
DatabaseHelper	x	x	x	x	x	x	x	x

The **Controller** class will be the interconnected with all other classes. This is the only domain concept that's itself because everything is directly accessible through this class.

The **Guest** object will keep track guest account information on luggage status, reservations, check in/out, and account status. This is related to UserVerification because it is needed to check with the database if the email has already been used before. It is also part of the UserManagement concept because all their account information and object fields will be stored in the database.

The **GuestControl** will be responsible for checking proper login request for each guest and marking them as currently logged in and out. Additionally, GuestControl will also be in charge of adding and removing guests from the database.

The **Admin** object will be directly from the domain concepts AdminControl but also interacts with other concepts. This will be the platform in which hotel admin will be able to login to their own system. This under the AdminControl as the administrators/hotel staff (i.e bellboy, valet, concierge) will all be the same object, but differentiated by their own separate duties/roles within the system. This distinction between guest and administrator will be necessary for functions within the app.

The **RoomObject** will be related to RoomTracking, and RoomControl. Each Room will be its own object once the guest checks into the room. The object will be used to manage the different parts of the room, such as room key, maintenance request and reversed dates. RoomTracking concept is meant to analyze all the types of rooms available on specific dates, so this will use the Room object data to accomplish this. And the RoomControl concept allows the guest to make a reservation and check-in, so the Room object data will be updated accordingly.

The **ReservationControl** is for retrieving information of reservations from guests and identifying the available rooms for available days. This interacts with the RoomControl and RoomTracking to determine the available rooms at the time of the request. Once the reservation is confirmed, the data is sent back and updated in the concepts.

The **DatabaseHelper** is a separate object to handle all the SQL statements and directly change the stored data. Within this class, there will be methods for each object to interact with, for example, having an addUser() methods to run SQL that add a user to the UserTable. Therefore, the DatabaseHelper will interact with each concept because each concept requires data management.

System Architecture and System Design

Architectural Style

Communication

The basis for our RUstaying app lies upon the many functionalities that we provide to our guests through hotel automation. Through Service-Oriented Architecture we are building each of these functions independent of each other. These implemented services communicate messages with our app through our formally defined XML schemas in Android Studio. This allows us to reuse code and simplify the management of said code. SOA helps keep our code well-defined and self-contained with its various functions because at the end, we need robust capability for the app to be able to communicate with an end user and other entities like databases or another user with admin privileges. An advantage of using this architecture is that if we were to scale up or scale down on functions, the rest of the functions would still be just as efficient since they are self-reliant.

Deployment

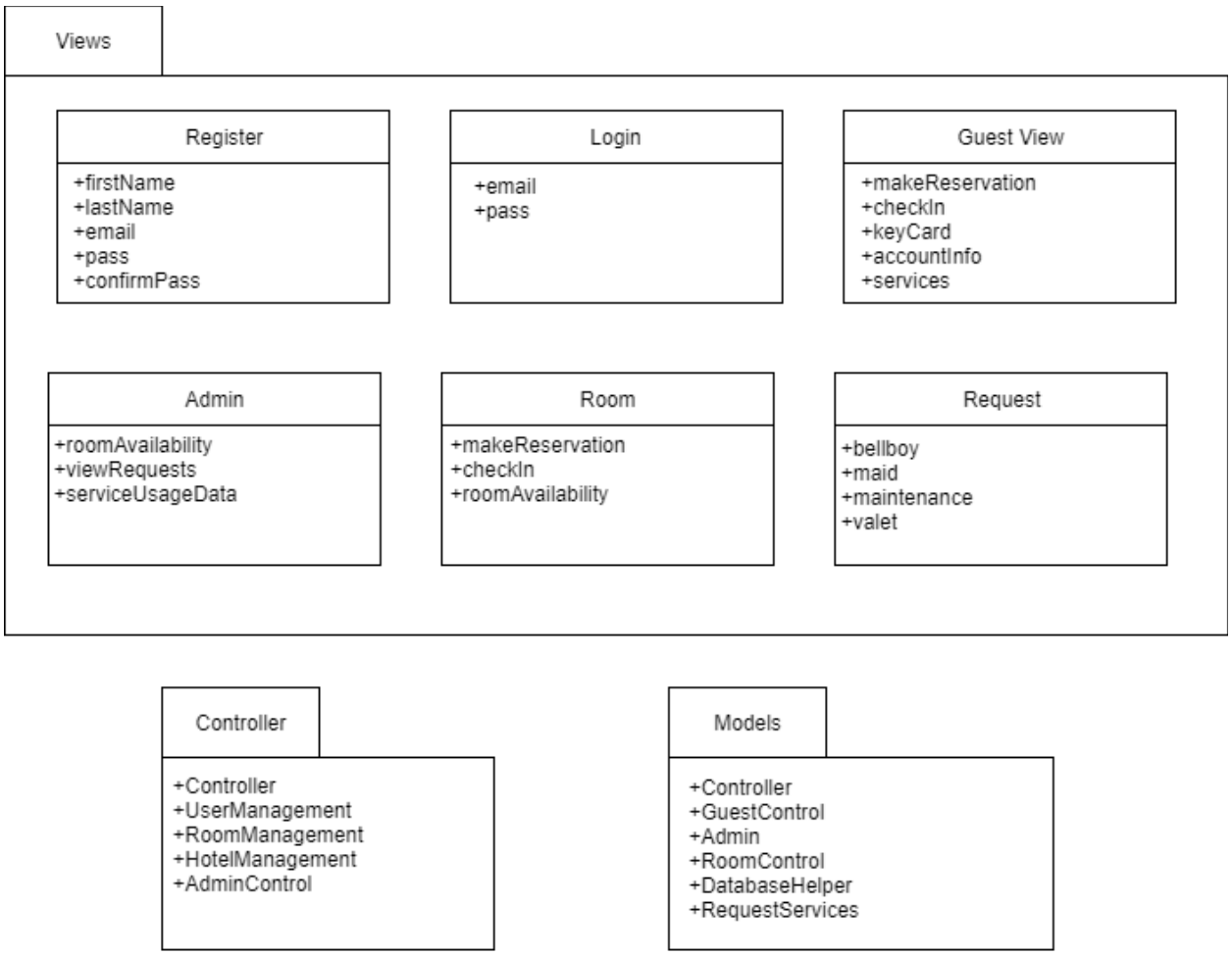
Current version of app uses SQLite. For the deployment of our app, once completely implemented, ideally we would need a database which is accessible through a cloud for when the Model needs to load/store data. This type of interface which is essentially a client-service communication so as a result we are using a client-server model. When there are multiple platforms supporting the app, such functionality for the interface needs to be consistent across all devices so having this model gives us the most efficiency. Ultimately, we rely on these user-server requests and this model gives us the most efficiency and scalability

Structure

Using MVC(Model-View-Controller) for Object Oriented Architecture

In our case, the controller component is what initiates a function, manipulates data in the Model, and returns some values and displays the data for end user to view. Generically, once the user has some input in the UI, the controller will handle the logic and call the function requested. The Model, which in this case are the independent functions(may or may not use database) will load/store data. Here is where all the classes which the RUstaying team has written will come into play. Once controller has received confirmation that the function has completed partially/completed (app might need more input from UI for complex functions), controller will ask View to update for further instructions.

Identifying Subsystems



Persistent Data Storage

In order to keep track of data for long periods of time, we must use persistent data storage. Our app uses SQLite to store data obtained from the user. SQlite is a relational database management system. This allows you to have multiple tables that contain related data and that data is linked by a common value that is stored in both tables. The database stores sensitive information about each guest, such as username, password, email, etc. It is crucial to keep track of such fields for long periods of time as they are required to login. The database also keeps track of reservation details, maintenance requests, digital room keys for each guest. The statuses of each requests are also stored. These fields should be persistent to ensure that each request is being processed and completed.

To store persistent data, we will be creating database tables such as User Table, Room Table, and Services Table. Here is short description of each table and its use of persistent data:

```
CREATE TABLE ‘User’(
        ‘FirstName’ TEXT,
        ‘LastName’ TEXT,
        ‘guestID’ INT,
        ‘userPassword’ TEXT,
        ‘userEmail’ TEXT,
        ‘accountCreation’ DATE,
        ‘accountStatus’ TEXT,
        ‘hasReservation’ BOOLEAN,
        ‘checkedIn’ BOOLEAN,
        ‘loggedIn’ BOOLEAN,
        ‘roomKeyID’ TEXT
)
```

Description: As guestID, userPassword, and userEmail is required to login, this information must be persistent. The system should also keep track of the user first and last name. accountCreation, accountStatus, hasReservation, checkIn, loggedIn are also all important fields to keep persistent as some of these booleans need to be TRUE for the user to activate certain services. roomKeyID should most definitely be persistent as the user would not be able to enter the room without it each time.

```
CREATE TABLE ‘Room’(
        ‘roomId’ INT,
        ‘roomType’ TEXT,
        ‘isAvailable’ BOOLEAN,
        ‘nextReservation’ DATE,
        ‘roomService’ TEXT,
        ‘maintenanceRequest’ TEXT
)
```

Description: To keep track of each room and its availability, it is vital that we keep fields such as roomId, roomType, isAvailable and nextReservation persistent. Without this persistent data, the staff would not know which rooms are currently available or will be in the future.

```
CREATE TABLE ‘Admin’(
        ‘adminId’ INT,
        ‘adminEmail’ TEXT,
```

‘adminPassword’ TEXT,
‘adminType’ TEXT
)

Description: Since the staff runs the hotel, there is user information available to them. To make sure only an admin can access such information, adminId, adminEmail, adminPassword are kept persistent fields. No one other than a respective adminType should be able to access certain data and services.

Global Control Flow

Execution Order

This app is procedure-driven up until after registration and login, meaning every user will go through the same steps every time. The rest following is event-driven which means the user determines what the app is used for and which features they want to use. Upon opening the app, all users must either log in or if they do not have an account, register and then log in. Each user can use the app whenever they want and in order of whatever features they would like after being logged in. For example, once the user is logged in, they can choose to do anything like book a room, request a service, or even logout.

Time Dependency

This app is a real-time dependent system that refreshes its information every 5 seconds so that the user and admin can both have the most updated version on room vacancies, check outs, check ins and if a maintenance/room service request has been completed. Every function in the app is either dependent on what the users or hotel staff is doing at any given moment. For example, a guest will check into a room for a set amount of days in real time and hotel staff will be completing services such as room service in real time and the app will be updated.

Concurrency

Our app will not be using multithreading or multiprocessing. The user interface is event-controlled and each action only requires linear execution. There is no need for concurrency because the user can only initiate one action at a time. Synchronization is not directly used within our app but will be used by the database to store information properly. Because multiple users can be using the app at the same time, there could be multiple requests to get or update the data stored in the database. This, however, is managed directly by SQLite in its implementation. We will not need to worry about synchronization because SQLite will be managing concurrent data requests at the same time to ensure the data is not corrupted.

Hardware Requirements

As a whole, the majority of computational resources are used on the server-side with the use of Android Studio, SQLite database, and a Java backend which can be accessed by Android phones. The system replies on the use of Internet Connection in order to constantly pull data and verify the features of the application the user chooses to the admin and the employees of the hotel. This is also necessary in order to maintain consistency of uses for all of the users and maintain a constant updated version of room status, service status, check out/check in, etc. The internet is also required for the concierge feature of the app in order to help the user find nearby activities/restaurants or allow any questions of the user to be sent to a concierge themselves to answer. Due to the fact that we are using Android Studio, we are able to customize the app's layout, allowing us to be flexible with the resolution, scaling size and disk space. Android Studio allows us to create alternate layouts as well in order to adjust to how the user is holding their phone while viewing the app- horizontally or vertically. Users will be able to access the app through most Android smartphones or simulators that act as one.

Project Management

These are milestone updates since Report 2 Part 1 was submitted. The completed milestones are removed from the list and a couple other milestones are now in progress. Unfortunately we were not able to complete any new milestones yet, but we have delegated the work accordingly within our group and a few of the milestones have made progress. We also adjusted the deadlines slightly based on some difficulties we faced implementing the milestone. These are milestones we would like to have completed by Demo 1 and we are on track to complete all of them.

Updates on Milestones

1. Design all other pages the user can navigate to in the app
    a. Look at the Navigation Tree created in report 1 and start to create all those pages in Android Studio. Also, maintain consistency with the current pages already created and follow same color theme. This is just an user interface to be able to implement functionality with later on.
    b. Update: We created a couple new pages and have implemented them into the app for far.
    c. Status: In Progress
    d. Due: 3/12/19
2. Administrator Account Login
    a. Create preset email addresses and passwords that indicate a manager account. The manager account will have a slightly different user interface but most pages should be reused from the guest user interface. This milestone is to design the new XML pages specifically for an admin and to implement the functionality for an admin to login
    b. Update: The Admin object was created and we have to make preset email addresses and passwords and store them into the database.
    c. Status: In Progress
    d. Due: 3/12/19
3. Check availability of rooms
    a. Create a database table with all the types of rooms available at the hotel, each with an available/unavailable flag. In the table, we will also have columns for the duration the specific room is being used for. The table is collection of all the rooms and the exact dates they are reserved for. This functionality is necessary for other use cases and keeping track of data.
    b. Update: The Room object has been created and a database table of all the rooms available with their corresponding types. We have to implement the database methods to update the fields of the Room object in the database.

    c. **Status: In Progress**
    d. **Due: 3/8/19**
4. Implement Make a Reservation functionality
    a. One of the most used features, is to make a reservation at the hotel. This corresponds to Use Case 4. To be able to implement this use case, we will need to use the functionality of the previous milestone. The user will be prompted to enter information about the room and the duration of their stay with exact dates. When a new reservation request comes in, we will check the table of all rooms and only present the guest with the rooms that have the "available" flag. This table will be constantly updated as guests make reservations, so that no unavailable rooms are accidently shown to guests. When a reservation is confirmed, the information will be stored in the UserTable under the appropriate guest. This milestone requires a few steps and is expected to be a little challenging so we allocated more time for it.
    b. **Update:** Waiting on Milestone 4 to be completed.
    c. **Status: Not Started**
    d. **Due: 3/15/19**
5. Allow guest to check-in for their stay
    a. This milestone is also dependent on the previous one, as a guest needs to have a reservation in order to check-in. This is a relatively simple milestone, as we only have to check if the guest has a reservation for a room. If the guest successfully checks into a room, then the room is going to marked with a flag to indicate that it is now in use.
    b. **Update:** Waiting on Milestone 5 to be completed.
    c. **Status: Not Started**
    d. **Due: 3/17/19**
6. Feedback form
    a. This case occurs when the guest has ended their stay at the hotel. We know the guest's stay has ended because when we made a reservation, they specified a start and end date. So the feedback form will be prompted when it's the guests last day at the hotel. The main part of this milestone is to design the form and prompt the guest on the last day, so it is relatively simple, but dependent on completing other milestones
    b. Update: The XML page has been created for the feedback form. The functionality has also been integrated with the app. The only part remaining is to store the data into the database.
    c. Status: In Progress
    d. Due: 3/17/19
7. Frequently Asked Questions Interface

    a. This will be a static page with common questions about using the app and hotel services. The purpose of this page is to gives the user instructions on how the app works and some conveniences provided. This does not have much functionality so it should be relatively quick to implement.
    b. Update: None
    c. Status: Not Started
    d. Due: 3/8/19
8. Valet & Travel/ Bellboy Interface
    a. These two services are relatively similar in their functionality. The guest will request this service through the app and the staff will be notified of a new request made. The user will get a notification when the request is completed. We need to implement an interface that allows staff to give updates that will notify the guests of the status of their requests.
    b. Update: None
    c. Status: In Progress
    d. Due: 3/17/19
9. Debug and test implemented features
    a. The milestones listed above all due before the first demo (3/25). By meeting these due dates, we will have enough time afterwards to test our these features and fix any bugs. The purpose of this is to have a seamless software with enough functionality by the time of the first demo.
    b. **Update:** Continuing testing as each new feature is being implemented.
    c. **Status: In Progress**
    d. **Due: 3/24/19**

References

SQLite - Create a Relationship
https://www.quackit.com/sqlite/tutorial/create_a_relationship.cfm
Advantages of Service Oriented Architecture
https://techspirited.com/advantages-disadvantages-of-service-oriented-architecture-soa
10 Common software architectural patterns
https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

Android Architecture Patterns
https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6